# LLMs:
# PEFT, RLHF, RAG

Lecture 15

18-789

# Administrative

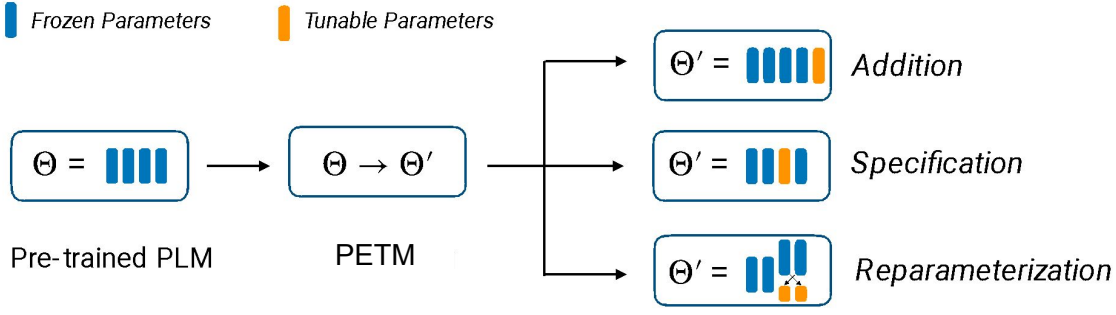
- Today
  - Recap: PEFT
  - Instruction Tuning, RLHF
  - Reasoning Model
  - Retrieval-Augmented Training, RAG

# Why parameter-efficient tuning?

- Prompt engineering
  - Doesn’t always work
  - Tedious to find a good prompt
- Finetune the full LM
  - Expensive
  - Overfitting / catastrophic forgetting on small datasets
  - Need to store one full set of model weights per task
- PEFT
  - Rather than finetuning the entire model, we finetune only small amounts of weights.

# PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- Specification: What if we pick a small subset of the parameters of the neural network and just tune those?
- Reparameterization: What if we re-parameterize the model into something that is more efficient to train?



# Prompt Tuning

**The Power of Scale for Parameter-Efficient Prompt Tuning**

**Brian Lester*** **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com

**Abstract**

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. Our method can be seen

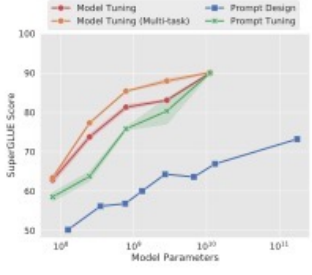
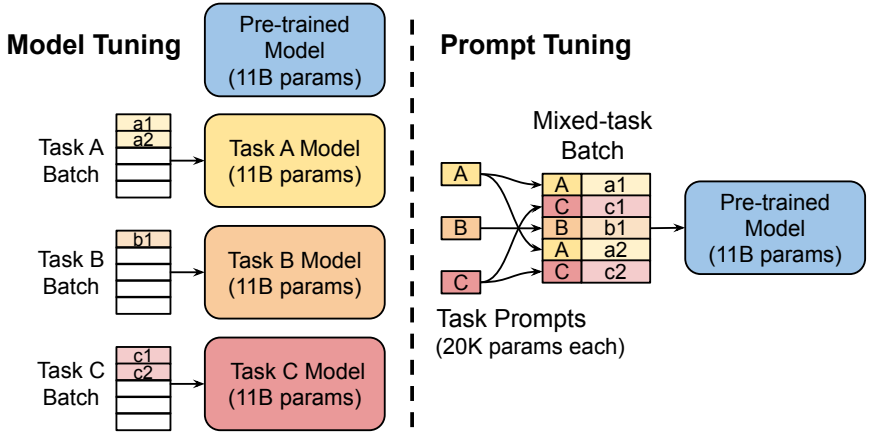



Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.



# Prompt Tuning

**The Power of Scale for Parameter-Efficient Prompt Tuning**

**Brian Lester*** **Rami Al-Rfou** **Noah Constant**
Google Research
{brianlester,rmyeid,nconstant}@google.com

**Abstract**

In this work, we explore "prompt tuning," a simple yet effective mechanism for learning "soft prompts" to condition frozen language models to perform specific downstream tasks. Unlike the discrete text prompts used by GPT-3, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples. Our end-to-end learned approach outperforms GPT-3's few-shot learning by a large margin. More remarkably, through ablations on model size using T5, we show that prompt tuning becomes more competitive with scale: as models exceed billions of parameters, our method "closes the gap" and matches the strong performance of model tuning (where all model weights are tuned). This finding is especially relevant because large models are costly to share and serve and the ability to reuse one frozen model for multiple downstream tasks can ease this burden. Our method can be seen

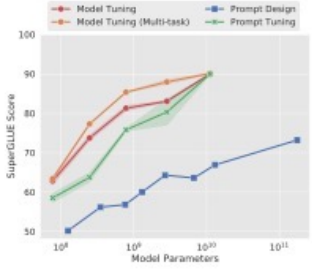



Figure 1: Standard **model tuning** of T5 achieves strong performance, but requires storing separate copies of the model for each end task. Our **prompt tuning** of T5 matches the quality of model tuning as size increases, while enabling the reuse of a single frozen model for all tasks. Our approach significantly outperforms few-shot **prompt design** using GPT-3. We show mean and standard deviation across 3 runs for tuning methods.

- Finetune T5 to act a bit more like a traditional language model.
  - This only needs to be done once, and empirically makes prompt tuning working better.
  - This is probably because the span-corruption objective T5 was originally trained with isn't amenable to prompting.
- Freeze the weights of T5. Set the first k input embeddings to be learnable.
  - k is a hyperparameter up to the choice of the implementer.
- Initialize the k learnable embeddings. Some options include:
  - Random initialization
  - Initialize to values drawn from the vocabulary embedding matrix.
- Train on your task specific data

# Prefix Tuning

**Prefix-Tuning: Optimizing Continuous Prompts for Generation**

**Xiang Lisa Li**
Stanford University
xlisali@stanford.edu

**Percy Liang**
Stanford University
pliang@cs.stanford.edu

**Abstract**

Fine-tuning is the de facto way of leveraging large pretrained language models for downstream tasks. However, fine-tuning modifies all the language model parameters and therefore necessitates storing a full copy for each task. In this paper, we propose prefix-tuning, a lightweight alternative to fine-tuning for natural language generation tasks, which keeps language model parameters frozen and instead optimizes a sequence of *continuous task-specific* vectors, which we call the *prefix*. Prefix-tuning draws inspiration from prompting for language models, allowing subsequent tokens to attend to this prefix as if it were "virtual tokens". We apply prefix-tuning to GPT-2 for table-to-text generation and to BART for summarization. We show that by modifying only 0.1% of the parameters, prefix-tuning obtains comparable performance in the full data setting, outperforms fine-tuning in low-data settings, and extrapolates better to examples with topics that

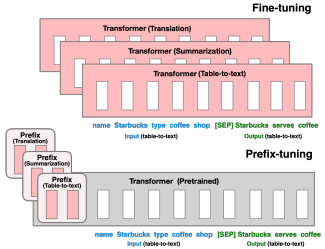
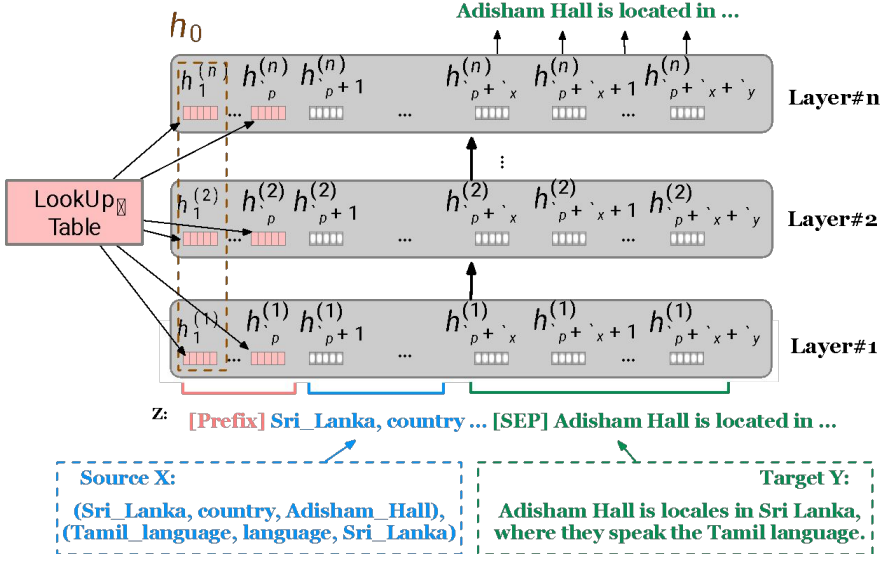



Figure 1: Fine-tuning (top) updates all LM parameters (the red Transformer box) and requires storing a full model copy for each task. We propose prefix-tuning (bottom), which freezes the LM parameters and only optimizes the prefix (the red prefix blocks). Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient. Note that each vertical block denote transformer activations at one time step.



# Advantages / Disadvantages

- The learned embeddings tend to be relatively small, just a few megabytes or less.
  - It is cheap to keep around one set of embeddings per task.
- The pre-trained LLM can be loaded into memory (such as on a server), and at inference time, the appropriate task-specific embeddings can be passed in.
  - Example use case: User customization

- In practice, these methods tend to converge significantly slower than full parameter fine-tuning.
- Unclear what the best prefix length is for any particular task.
  - Every sequence position you "spend" on the prefix is one less you have for your actual task.
- Learned embeddings are not very interpretable.

# Adapters

## Parameter-Efficient Transfer Learning for NLP

Neil Houlsby [1] Andrei Giurgiu [1 *] Stanisław Jastrzębski [2 *] Bruna Morrone [1] Quentin de Laroussilhe [1]
Andrea Gesmundo [1] Mona Attariyan [1] Sylvain Gelly [1]

### Abstract

Fine-tuning large pre-trained models is an effective transfer mechanism in NLP. However, in the presence of many downstream tasks, fine-tuning is parameter inefficient: an entire new model is required for every task. As an alternative, we propose transfer with adapter modules. Adapter modules yield a compact and extensible model; they add only a few trainable parameters per task, and new tasks can be added without revisiting previous ones. The parameters of the original network remain fixed, yielding a high degree of parameter sharing. To demonstrate adapter's effectiveness, we transfer the recently proposed BERT Transformer model to 26 diverse text classification tasks, including the GLUE benchmark. Adapters attain near state-of-the-art performance, whilst adding only a few parameters per task. On GLUE, we attain within 0.4% of the performance of full fine-tuning, adding only 3.6% parameters per task. By contrast, fine-tuning trains 100% of the parameters per task.[1]

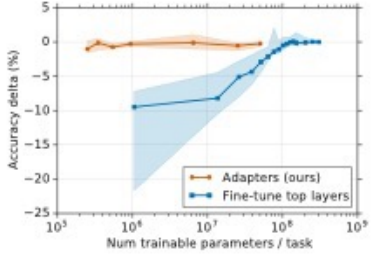
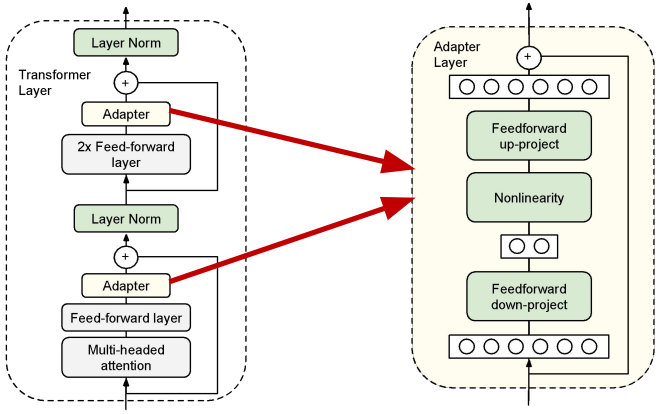



*Figure 1.* Trade-off between accuracy and number of trained task-specific parameters, for adapter tuning and fine-tuning. The y-axis is normalized by the performance of full fine-tuning, details in Section 3. The curves show the 20th, 50th, and 80th performance percentiles across nine tasks from the GLUE benchmark. Adapter-based tuning attains a similar performance to full fine-tuning with two orders of magnitude fewer trained parameters.



# Compacters

## COMPACTER: Efficient Low-Rank Hypercomplex Adapter Layers

**Rabeeh Karimi Mahabadi**
EPFL University, Idiap Research Institute
rabeeh.karimi@idiap.ch

**James Henderson**
Idiap Research Institute
james.henderson@idiap.ch

**Sebastian Ruder**
DeepMind
ruder@google.com

**Abstract**

Adapting large-scale pretrained language models to downstream tasks via fine-tuning is the standard method for achieving state-of-the-art performance on NLP benchmarks. However, fine-tuning all weights of models with millions or billions of parameters is sample-inefficient, unstable in low-resource settings, and wasteful as it requires storing a separate copy of the model for each task. Recent work has developed *parameter-efficient* fine-tuning methods, but these approaches either still require a relatively large number of parameters or underperform standard fine-tuning. In this work, we propose COMPACTER, a method for fine-tuning large-scale language models with a better trade-off between task performance and the number of trainable parameters than prior work. COMPACTER accomplishes this by building on top of ideas from adapters, low-rank optimization, and parameterized hypercomplex multiplication layers.

Specifically, COMPACTER inserts task-specific weight matrices into a pretrained model's weights, which are computed efficiently as a sum of Kronecker products between shared "slow" weights and "fast" rank-one matrices defined per COMPACTER layer. By only training $0.047\%$ of a pretrained model's parameters, COMPACTER performs on par with standard fine-tuning on GLUE and outperforms standard fine-tuning on SuperGLUE and low-resource settings. Our code is publicly available at https://github.com/rabeehk/compacter.

- **Compacters** are an extension of adapters which aim to make the technique even more efficient.
- Adapters are standard fully connected layers.
  - Linear project to lower dimension followed by nonlinearity followed by projection back up to original dimension.
  - $y = \boldsymbol{W_2}\ \mathrm{GELU}(\boldsymbol{W_1 x} + \boldsymbol{b_1}) + \boldsymbol{b_2}$
- The compacter replaces the fully connected layer with a. parameterized hypercomplex multiplication layer.
  - Each $\boldsymbol{W}$ is learned as a sum of $n$ Kronecker products
  - $n$ is a user-specified hyperparameter.
- Compacters reduce the number of parameters in the adapter layer to $1/n$ without harming the performance.

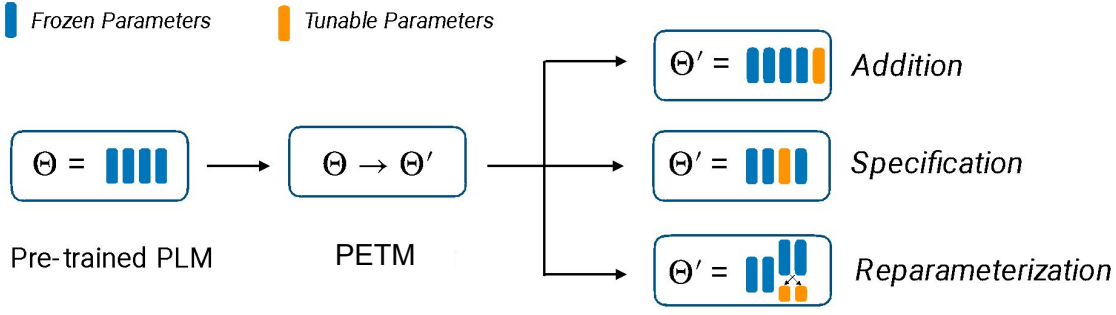
# Advantages / Disadvantages

- Have been shown to be quite effective in multi-task settings.
  - There are methods for training task-specific adapters and then combining the to leverage the cross-task knowledge• Tend to be faster to tune than full model finetuning.
- Possibly more robust to adversarial perturbations of the tuning data than full model finetuning.

- Adding in new layers means making inference slower.
- It also makes the model bigger (possibly harder to fit on available GPUs).
- Adapter layers need to be processed sequentially at inference time, which can break model parallelism.

# PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- **Specification: What if we pick a small subset of the parameters of the neural network and just tune those?**
- Reparameterization: What if we re-parameterize the model into something that is more efficient to train?



# Layer Freezing

**BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models**

**Elad Ben-Zaken**[1] **Shauli Ravfogel**[1,2] **Yoav Goldberg**[1,2]
[1]Computer Science Department, Bar Ilan University
[2]Allen Institute for Artificial Intelligence
{benzakenelad, shauli.ravfogel, yoav.goldberg}@gmail.com

**Abstract**

We introduce BitFit, a sparse-finetuning method where only the bias-terms of the model (or a subset of them) are being modified. We show that with small-to-medium training data, applying BitFit on pre-trained BERT models is competitive with (and sometimes better than) fine-tuning the entire model. For larger data, the method is competitive with other sparse fine-tuning methods. Besides their practical utility, these findings are relevant for the question of understanding the commonly-used process of finetuning: they support the hypothesis that finetuning is mainly about exposing knowledge induced by language-modeling training, rather than learning new task-specific linguistic knowledge.

3. The changed parameters are both isolated and localized across the entire parameter space.
4. For small to medium training data, changing only these parameters reaches the same task accuracy as full fine-tuning, and sometimes even improves results.

Specifically, we show that freezing most of the network and **fine-tuning only the bias-terms** is surprisingly effective. Moreover, if we allow the tasks to suffer a small degradation in performance, we can fine-tune only two bias components (the "query" and "middle-of-MLP" bias terms), amounting to half of the bias parameters in the model, and only 0.04% of all model parameters.

This result has a large practical utility in de-

- Only tune the bias terms and final classification layer (if doing classification)
- Recall the equations for multi-head attention

$$\mathbf{Q}^{m,\ell}(\mathbf{x}) = \mathbf{W}_q^{m,\ell}\mathbf{x} + \mathbf{b}_q^{m,\ell}$$
$$\mathbf{K}^{m,\ell}(\mathbf{x}) = \mathbf{W}_k^{m,\ell}\mathbf{x} + \mathbf{b}_k^{m,\ell}$$
$$\mathbf{V}^{m,\ell}(\mathbf{x}) = \mathbf{W}_v^{m,\ell}\mathbf{x} + \mathbf{b}_v^{m,\ell}$$

  - $\ell$ is the layer index
  - $m$ is the attention head index
  - Only the bias terms (shown in red) are updated.

# DiffPruning

**Parameter-Efficient Transfer Learning with Diff Pruning**

**Demi Guo**
Harvard University
dguo@college.harvard.edu

**Alexander M. Rush**
Cornell University
arush@cornell.edu

**Yoon Kim**
MIT CSAIL
MIT-IBM Watson AI
yoonkim@mit.edu

**Abstract**

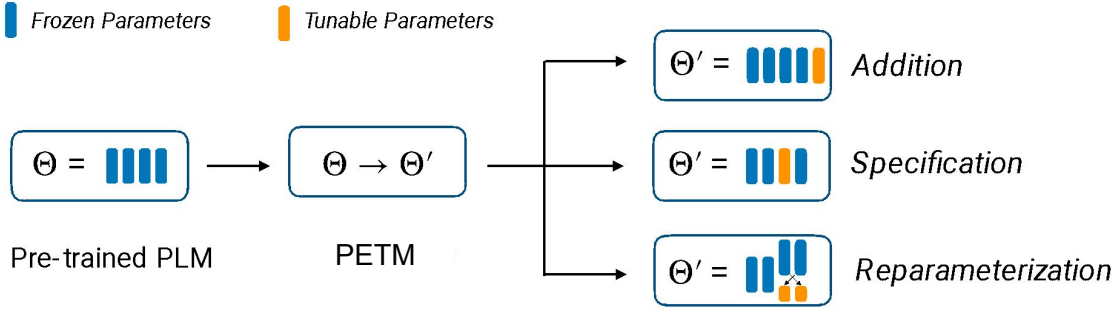
The large size of pretrained networks makes them difficult to deploy for multiple tasks in storage-constrained settings. *Diff pruning* enables parameter-efficient transfer learning that scales well with new tasks. The approach learns a task-specific "diff" vector that extends the original pretrained parameters. This diff vector is adaptively pruned during training with a differentiable approximation to the $L_0$-norm penalty to encourage sparsity. As the number of tasks increases, diff pruning remains parameter-efficient, as it requires storing only a small diff vector for each task. Since it does not require access to all tasks during training, it is attractive in on-device deployment settings where tasks arrive in stream or even from different providers. Diff pruning can match the performance of finetuned baselines on the GLUE benchmark while only modifying 0.5% of the pretrained model's pa-

A popular approach to parameter-efficiency is to learn smaller compressed models for each task (Gordon et al., 2020; Sajjad et al., 2020; Zhao et al., 2020; Sanh et al., 2020). Such approaches face a steep sparsity/performance tradeoff and keep a substantial amount of nonzero parameters per task (e.g. 10%-30%). Multi-task learning and feature-based transfer allow for more parameter-efficient transfer learning per task (Liu et al., 2019b; Clark et al., 2019; Stickland & Murray, 2019; Reimers & Gurevych, 2019). These methods train a small number of additional parameters (e.g. a linear layer) on top of a shared model. However, multi-task learning generally requires access to all tasks during training to prevent catastrophic forgetting (French, 1999), while feature-based transfer learning (e.g. based on task-agnostic sentence representations) is typically outperformed by finetuning (Howard & Ruder, 2018).

- In prior methods we discussed, the choice of what parameters to freeze and what parameters to tune was done manually.
- Why not learn this instead?
- Main idea:
  - For each parameter, finetune a learnable "delta" which gets added to the original parameter value.
  - Use an $L_0$-norm penalty to encourage sparsity in the deltas.

# PEFT

- Addition: What if we introduce additional trainable parameters to the neural network and just train those?
- Specification: What if we pick a small subset of the parameters of the neural network and just tune those?
- **Reparameterization: What if we re-parameterize the model into something that is more efficient to train?**



# Intuition

- Finetuning has a low **intrinsic dimension**, that is, the minimum number of parameters needed to be modified to reach satisfactory performance is not very large.
- This means, we can reparameterize a subset of the original model parameters with low-dimensional proxy parameters, and just optimize the proxy.

- An objective function's intrinsic dimension measures the minimum number of parameters needed to reach a satisfactory solution to the objective.
- Can also be thought of as the lowest dimensional subspace in which one can optimize the original objective function to within a certain level of approximation error.

# LoRA

## LoRA: Low-Rank Adaptation of Large Language Models

**Edward Hu*** **Yelong Shen*** **Phillip Wallis** **Zeyuan Allen-Zhu**
**Yuanzhi Li** **Shean Wang** **Lu Wang** **Weizhu Chen**
Microsoft Corporation
{edwardhu, yeshe, phwallis, zeyuana,
yuanzhil, swang, luw, wzchen}@microsoft.com
yuanzhil@andrew.cmu.edu
(Version 2)

### Abstract

An important paradigm of natural language processing consists of large-scale pre-training on general domain data and adaptation to particular tasks or domains. As we pre-train larger models, full fine-tuning, which retrains all model parameters, becomes less feasible. Using GPT-3 175B as an example – deploying independent instances of fine-tuned models, each with 175B parameters, is prohibitively expensive. We propose **Lo**w-**R**ank **A**daptation, or LoRA, which freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, greatly reducing the number of trainable parameters for downstream tasks. Compared to GPT-3 175B fine-tuned with Adam, LoRA can reduce the number of trainable parameters by 10,000 times and the GPU memory requirement by 3 times. LoRA performs on-par or better than fine-tuning in model quality on RoBERTa, DeBERTa, GPT-2, and GPT-3, despite having fewer trainable parameters, a higher training throughput, and, unlike adapters, *no additional inference latency*. We also provide an empirical investigation into rank-deficiency in language model adaptation, which sheds light on the efficacy of LoRA. We release a package that facilitates the integration of LoRA with PyTorch models and provide our implementations and model checkpoints for RoBERTa, DeBERTa, and GPT-2 at `https://github.com/microsoft/LoRA`.

- Intuition: It's not just the model weights that are low rank, *updates* to the model weights are also low-rank.
- LoRA freezes the pretrained model weights and injects trainable rank decomposition matrices into each layer.
- Like DiffPruning, we are learning a delta to apply to each weight. In the case of LoRA, this delta has been re-paramaterized to be lower dimension than the original model parameters.
- In practice, LoRA only adapts the attention weights and keeps the rest of the Transformer as-is.

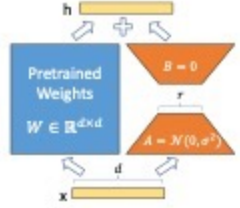


Figure 1: Our reparametrization. We only train $A$ and $B$.

# QLoRA

## QLoRA: Efficient Finetuning of Quantized LLMs

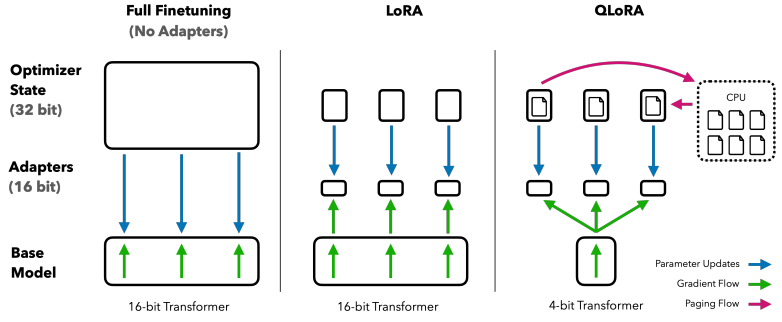
**Tim Dettmers*** **Artidoro Pagnoni*** **Ari Holtzman**

**Luke Zettlemoyer**

University of Washington
{dettmers,artidoro,ahai,lsz}@cs.washington.edu

### Abstract

We present QLoRA, an efficient finetuning approach that reduces memory usage enough to finetune a 65B parameter model on a single 48GB GPU while preserving full 16-bit finetuning task performance. QLoRA backpropagates gradients through a frozen, 4-bit quantized pretrained language model into Low Rank Adapters (LoRA). Our best model family, which we name **Guanaco**, outperforms all previous openly released models on the Vicuna benchmark, reaching 99.3% of the performance level of ChatGPT while only requiring 24 hours of finetuning on a single GPU. QLoRA introduces a number of innovations to save memory without sacrificing performance: (a) 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights (b) Double Quantization to reduce the average memory footprint by quantizing the quantization constants, and (c) Paged Optimizers to manage memory spikes. We use QLoRA to finetune more than 1,000 models, providing a detailed analysis of instruction following and chatbot performance across 8 instruction datasets, multiple model types (LLaMA, T5), and model scales that would be infeasible to run with regular finetuning (e.g. 33B and 65B parameter models). Our results show that QLoRA finetuning on a small high-quality dataset leads to state-of-the-art results, even when using smaller models than the previous SoTA. We provide a detailed analysis of chatbot performance based on both human and GPT-4 evaluations showing that GPT-4 evaluations are a cheap and reasonable alternative to human evaluation. Furthermore, we find that current chatbot benchmarks are not trustworthy to accurately evaluate the performance levels of chatbots. A lemon-picked analysis demonstrates where **Guanaco** fails compared to ChatGPT. We release all of our models and code, including CUDA kernels for 4-bit training.[2]

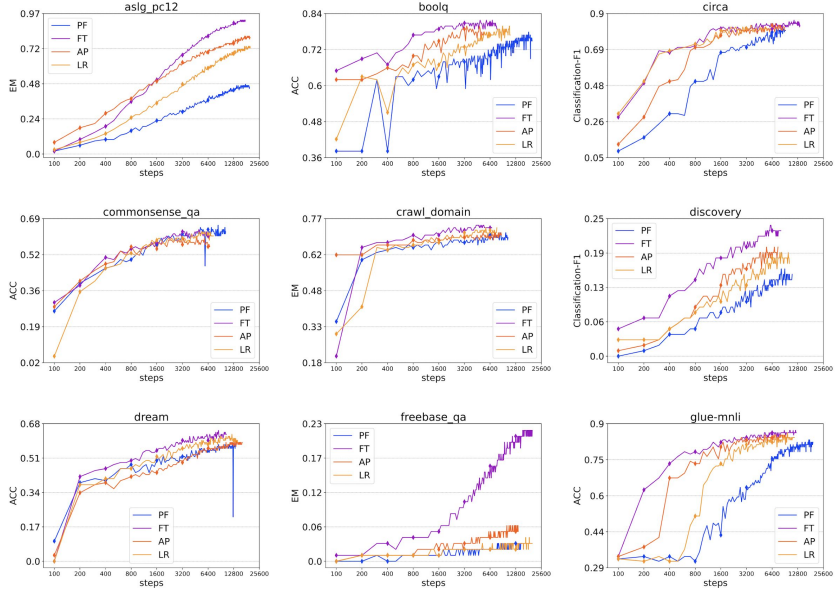


# Summary

- Training tends to be more memory-efficient, since we only need to calculate gradients and maintain optimizer state for a small number of parameters.
- These methods are faster to tune than standard full model finetuning.
- It is straight-forward to swap between tasks by swapping in and our just the tuned weights.

# Results

If you have the resources, full fine-tuning tends to work the best.




PF: prefix tuning

FT: full fine-tuning

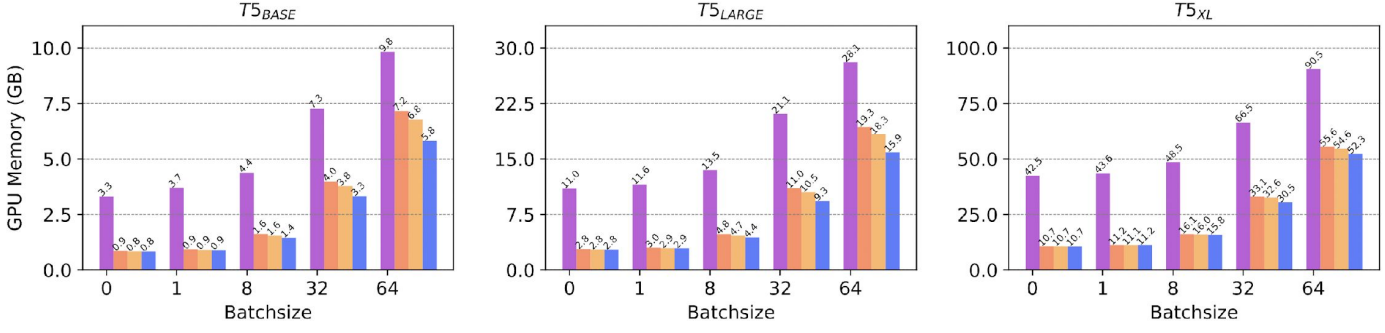
AP: adapter

LR: LoRA

**This survey overall found:**
Full fine-tuning >
LoRA >
Adapters >
Prefix Tuning >
Prompt Tuning
**In terms of performance.**

*Plots for many more tasks can be found in the paper.*

# Results




FT: full fine-tuning
AP: adapter
LR: LoRA
BF: BitFit
Prompt tuning and prefix tuning not included because they use the same amount of memory as full fine-tuning

# Why parameter-efficient tuning?

- Prompt engineering
  - Doesn’t always work
  - Tedious to find a good prompt
- Finetune the full LM
  - Expensive
  - Overfitting / catastrophic forgetting on small datasets
  - Need to store one full set of model weights per task
- **PEFT**
  - **Rather than finetuning the entire model, we finetune only small amounts of weights.**

# Finetuning vs. Instruction Tuning

Basic Finetuning:

- Build a model that is good at performing a single task

Instruction Tuning:

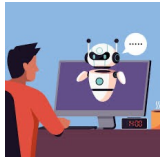
- Build a generalist model that is good at many tasks

What are --- Many tasks?

# What do we learn from pre-training

- Carnegie Mellon University is located in _____, Pennsylvania. [Trivia]
- I put ___ fork down on the table. [syntax]
- The woman walked across the street, checking for traffic over ___ shoulder. [coreference]
- I went to the ocean to see the fish, turtles, seals, and _____. [lexical semantics/topic]
- Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___. [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____ [some basic arithmetic; they don't learn the Fibonnaci sequence]

# What do we expect LLMs to do?



Agents?

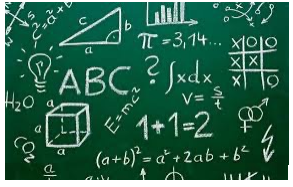
> ***Pat watches a demonstration of a bowling ball and a leaf being dropped at the same time in a vacuum chamber. Pat, who is a physicist, predicts that*** *the bowling ball and the leaf will fall at the same rate.*

Changing the last sentence of the prompt, we get:

> ***. . . Pat, who has never seen this demonstration before, predicts that*** *the bowling ball will fall to the ground first. This is incorrect. In a vacuum chamber, there is no air*

Language Models as Agent Models [Andreas, 2022 ]

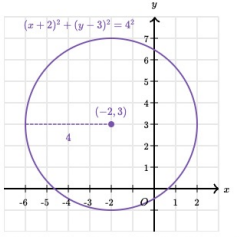
# What do we expect LLMs to do?




Math?

We can describe circles in the $xy$-plane using equations in terms of $x$ and $y$. Circle equations questions require us to understand the connection between these equations and the features of circles.

For example, the equation $(x+2)^2+(y-3)^2=4^2$ is graphed in the $xy$-plane below. It is a circle with a center at $(-2,3)$ and a radius of $4$.




https://www.khanacademy.org/test-prep/sat/x0a8c2e5f:untitled-652

# What do we expect LLMs to do?

Code?

```
// Determine whether the sentiment of text is positive
// Use a web service
async function isPositive(text: string): Promise<boolean> {
  const response = await fetch(`http://text-processing.com/api/sentiment/`, {
    method: "POST",
    body: `text=${text}`,
    headers: {
      "Content-Type": "application/x-www-form-urlencoded",
    },
  });
  const json = await response.json();
  return json.label === "pos";
}
```

https:// github.com/features/copilot

# What do we expect LLMs to do?

Medicine?

**Rapid and chronic ethanol tolerance are composed of distinct memory-like states in Drosophila**


## Abstract

Ethanol tolerance is the first type of behavioral plasticity and neural plasticity that is induced by ethanol intake, and yet its molecular and circuit bases remain largely unexplored. Here, we characterize three distinct forms of ethanol tolerance in male Drosophila: rapid, chronic, and repeated. Rapid tolerance is composed of two short-lived memory-like states, one that is labile and one that is consolidated. Chronic tolerance, induced by continuous exposure, lasts for two days, induces ethanol preference, and hinders the development of rapid tolerance through the activity of

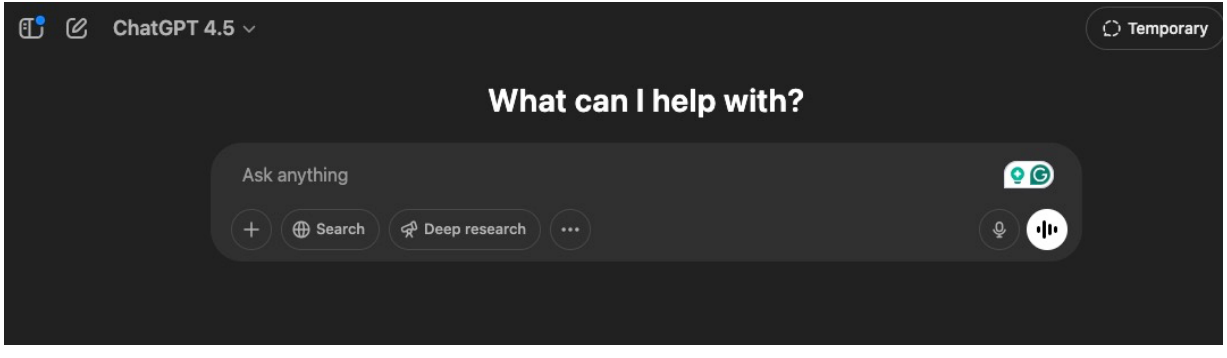

[Larnerd, 2023]

# LLMs as multitask assistants?

How do we go from:

Carnegie Mellon University is located in _____

to this?



# Language model ≠ assisting users

| | |
|---|---|
| PROMPT | *Explain the moon landing to a 6 year old in a few sentences.* |
| COMPLETION | GPT-3 |
| | `Explain the theory of gravity to a 6 year old.` |
| | `Explain the theory of relativity to a 6 year old in a few sentences.` |
| | `Explain the big bang theory to a 6 year old.` |
| | `Explain evolution to a 6 year old.` |

Language models are not *aligned* with user intent [Ouyang et al., 2022].

# Language model ≠ assisting users

| | |
|---|---|
| PROMPT | *Explain the moon landing to a 6 year old in a few sentences.* |
| COMPLETION | **Human**<br>A giant rocket ship blasted off from Earth carrying astronauts to the moon. The astronauts landed their spaceship on the moon and walked around exploring the lunar surface. Then they returned safely back to Earth, bringing home moon rocks to show everyone. |

Language models are not *aligned* with user intent [Ouyang et al., 2022]. Finetuning to the rescue!

# Instruction Finetuning and Alignment

Pre-trained language models should be “aligned” to follow user intentions. This is done through further finetuning.

Some possible alignment goals:

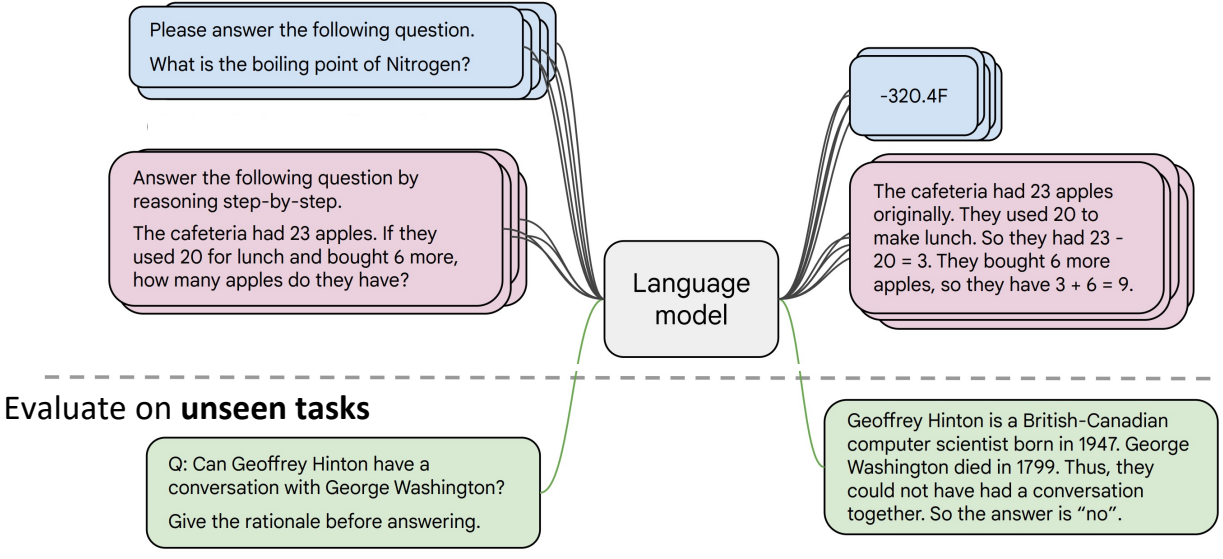
- The LLM should have a conversational interface.
- The LLM should be able to follow user instructions without complex prompt engineering.
- The LLMs’ generations should align with human values by being “helpful, honest and harmless.”

# LaMDA

- LaMDA was one of the first examples of alignment (from before “alignment” was a widely used term).
- Goal: create a chatbot which said sensible (but still interesting) things, and attempted to be factual, and followed safety guidelines.
- Pre-training procedure:
  - Decoder only language model trained for next token prediction
  - Trained on 2.97B documents and1.12B dialogs (acquired by scraping websites with conversational exchanges).

# Instruction Finetuning

**Collect examples** of (instruction, output) pairs across many tasks and finetune an LM




(1) Model size
8B -> 62B -> 540B

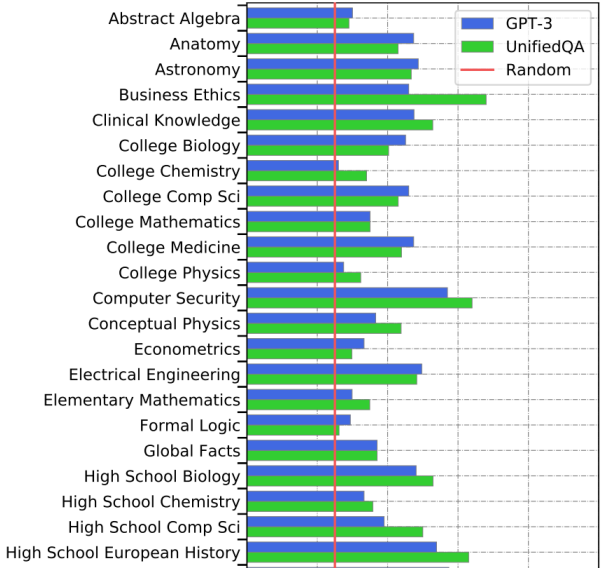
(2) More tasks
282 best

(3) Train with CoT
improves reasoning

[FLAN-T5; Chung et al., 2022]

# How to evaluate?

New benchmarks for measuring LM performance on 57 diverse *knowledge intensive* tasks

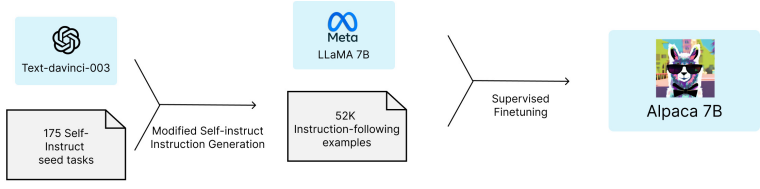


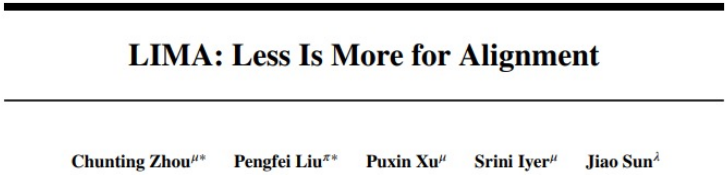

**Massive Multitask Language Understanding (MMLU)** [Hendrycks et al., 2021]

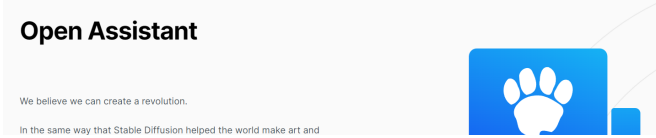

**BIG-Bench** [Srivastava et al., 2022]

# What have we learnt?

- You can generate data synthetically (from bigger LMs)
- You don’t need many samples to instruction tune
- Crowdsourcing can be pretty effective



# Limitations of Instruction finetuning

Instruction Tuning:

- Simple and straightforward, generalize to unseen tasks
- It's **expensive** to collect ground- truth data for tasks.
- Tasks like open-ended creative generation have no right answer.
- language modeling penalizes all token-level mistakes equally, but some errors are worse than others.
- Even with instruction finetuning, there a mismatch between the LM objective and the objective of "satisfy human preferences"!

Is it possible to explicitly attempt to satisfy human preferences?

# Reinforcement Learning from Human Feedback

E.g., Summarization task

SAN FRANCISCO, California (CNN) -- A magnitude 4.2 earthquake shook the San Francisco ... overturn unstable objects.

An earthquake hit San Francisco. There was minor property damage, but no injuries.

$s_1$

$R(s_1) = 8.0$

The Bay Area has good weather but is prone to earthquakes and wildfires.

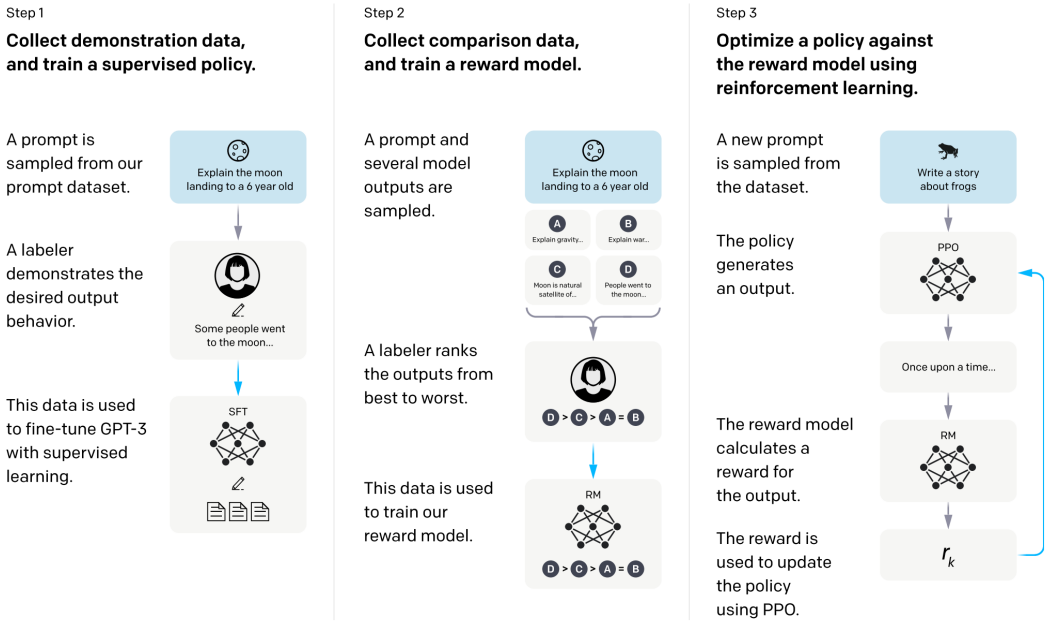
$s_2$

$R(s_2) = 1.2$

We want to maximize expected reward of samples from our LLM:

$$\mathbb{E}_{\hat{s} \sim p_\theta(s)}[R(\hat{s})]$$

# Reinforcement Learning from Human Feedback




Instruction tuning -> train reward model -> maximize reward

# Reinforcement Learning from Human Feedback

- The field of **reinforcement learning (RL)** has studied these (and related) problems for many years now [Williams, 1992; Sutton and Barto, 1998]
- Circa 2013: resurgence of interest in RL applied to deep learning, game-playing [Mnih et al., 2013]
- But the interest in applying RL to modern LMs is an even newer phenomenon [Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022]. **Why?**
  - RL w/ LMs has commonly been viewed as very hard to get right (still is!)
  - Newer advances in RL algorithms that work for large neural models, including language models (e.g. PPO; [Schulman et al., 2017])

# Reinforcement Learning from Human Feedback

How do we actually change our LM parameters $\theta$ to maximize this?

$$\mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})]$$

$$\theta_{t+1} \coloneqq \theta_t + \alpha \nabla_{\theta_t} \mathbb{E}_{\hat{s}\sim p_{\theta_t}(s)}[R(\hat{s})]$$

**How do we estimate this expectation??**

**What if our reward function is non-differentiable??**

**Policy gradient** methods in RL (e.g., REINFORCE; [Williams, 1992]) give us tools for estimating and optimizing this objective.

# Reinforcement Learning from Human Feedback

We want to obtain

(defn. of expectation) (linearity of gradient)

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \nabla_\theta \sum_s R(s) p_\theta(s) = \sum_s R(s) \nabla_\theta p_\theta(s)$$

Here we'll use a very handy trick known as the **log-derivative trick**. Let's try taking the gradient of $\log p_\theta(s)$

$$\nabla_\theta \log p_\theta(s) = \frac{1}{p_\theta(s)} \nabla_\theta p_\theta(s) \quad \Rightarrow \quad \nabla_\theta p_\theta(s) = p_\theta(s)\ \nabla_\theta \log p_\theta(s)$$

(chain rule)

Plug back in:

This is an expectation of this

$$\sum_s R(s) \nabla_\theta p_\theta(s) = \sum_s p_\theta(s) R(s) \nabla_\theta \log p_\theta(s)$$

$$= \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})\ \nabla_\theta \log p_\theta(\hat{s})]$$

Williams, 1992

# Reinforcement Learning from Human Feedback

- Now we have put the gradient "inside" the expectation, we can approximate this objective with Monte Carlo samples:

$$\nabla_\theta \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})] = \mathbb{E}_{\hat{s}\sim p_\theta(s)}[R(\hat{s})\,\nabla_\theta \log p_\theta(\hat{s})] \approx \frac{1}{m}\sum_{i=1}^{m} R(s_i)\,\nabla_\theta \log p_\theta(s_i)$$

This is why it's called **"reinforcement learning"**: we **reinforce** good actions, increasing the chance they happen again.

- Giving us the update rule:

$$\theta_{t+1} := \theta_t + \alpha \frac{1}{m}\sum_{i=1}^{m} R(s_i)\,\nabla_{\theta_t} \log p_{\theta_t}(s_i)$$

If $R$ is +++

Take gradient steps to maximize $p_\theta(s_i)$

If $R$ is ---

Take steps to minimize $p_\theta(s_i)$

This is **heavily simplified**! There is a *lot* more needed to do RL w/ LMs. **Can you see any problems with this objective?**

Williams, 1992

# Instruct GPT

- Goal: a decoder-only language model which can do zero-shot instruction following
- Pre-training procedure: same as the original GPT-3
- Fine-tuning data:
  - InstructGPT: two types of crowsourced data:
    - Crowdworkers answering prompts the way OpenAI wants the model
      - In contrast, previous instruction tuning approaches like T0 had humans developing natural language prompts for existing benchmark datasets.
    - Crowdworkers reading several LLM generations for a given prompt and ranking them.

# Problems?

**Problem 1:** human-in-the-loop is expensive!

**Solution:** instead of directly asking humans for preferences, **model their preferences** as a separate (NLP) problem! [Knox and Stone, 2009]

Train an LM $RM.s$ to predict human preferences from an annotated dataset, then optimize for $RM.$ instead.

**Problem 2:** human judgments are noisy and miscalibrated!

**Solution:** instead of asking for direct ratings, ask for **pairwise comparisons**, which can be more reliable [Phelps et al., 2015; Clark et al., 2018]

$$J_{RM}(\phi) = -\mathbb{E}_{(s^w, s^l)\sim D}\left[\log \sigma(RM_\phi(s^w) - RM_\phi(s^l))\right]$$

"winning" sample

"losing" sample

$s^w$ should score higher than $s^l$

# RLHF

Finally, we have everything we need:

- A pretrained (possibly instruction-finetuned) LM $p^{PT}(s)$
- A reward model $RM_\phi(s)$ that produces scalar rewards for LM outputs, trained on a dataset of human comparisons
- A method for optimizing LM parameters towards an arbitrary reward function.

Now to do RLHF:

- Initialize a copy of the model $p_\theta^{RL}(s)$ , with parameters $\theta$ we would like to optimize
- Optimize the following reward with RL:

$$R(s) = RM_\phi(s) - \underbrace{\beta \log \left( \frac{p_\theta^{RL}(s)}{p^{PT}(s)} \right)}$$

Pay a price when $p_\theta^{RL}(s) > p^{PT}(s)$

This is a penalty which prevents us from diverging too far from the pretrained model. In expectation, it is known as the **Kullback-Leibler** (**KL**) divergence between $p_\theta^{RL}(s)$ and $p^{PT}(s)$.

# Instruct GPT

Finetuning procedure

- Step 1: Finetuning LLM on high-quality instruction-style data
  - Built a dataset of prompts.
    - Some came from users of the OpenAI API, some were labeler-written.
  - Had labelers write responses for each prompt.
    - These responses are what OpenAI wants the model to say.
  - Fine-tuned the pre-trained LLM directly on these <prompt, response> pairs.
- Step 2: Training a reward model
  - On a second, larger dataset of prompts, used the finetuned model to generate a bunch of responses, then had human labelers rank several outputs for a given prompt.
  - Trained a reward model on this dataset of rankings to predict which model output labelers would prefer.
- Step 3: Further improving LLM with RL
  - Using reinforcement learning techniques, further fine-tuned the LLM to maximize the reward.

# ChatGPT: Instruction Finetuning + RLHF

**ChatGPT: Optimizing Language Models for Dialogue**

Note: OpenAI (and similar companies) are keeping more details secret about ChatGPT training (including data, training parameters, model size)—perhaps to keep a competitive edge…

## Methods

We trained this model using Reinforcem
Feedback (RLHF), using the same metho
slight differences in the data collection s
model using supervised fine-tuning: hur
conversations in which they played both
assistant. We gave the trainers access to
help them compose their responses. We
dataset with the InstructGPT dataset, w
dialogue format.

**(Instruction finetuning!)**

## Methods

To create a reward model for reinforcement learning, we needed to collect comparison data, which consisted of two or more model responses ranked by quality. To collect this data, we took conversations that AI trainers had with the chatbot. We randomly selected a model-written message, sampled several alternative completions, and had AI trainers rank them. Using these reward models, we can fine-tune the model using Proximal Policy Optimization. We performed several iterations of this process.

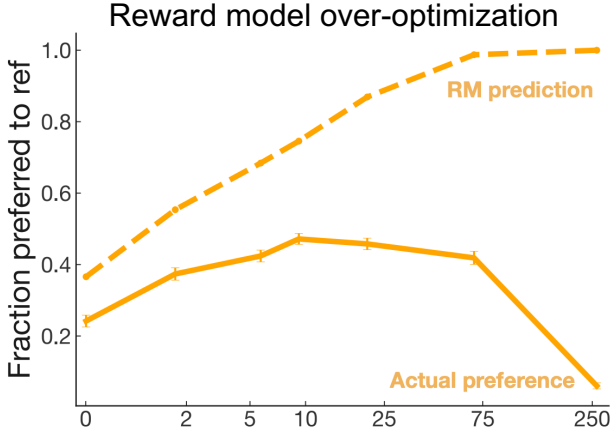
**(RLHF!)**

# Limitations of RL + Reward modeling

Human preferences are unreliable!

- "Reward hacking" is a common problem in RL
- Chatbots are rewarded to produce responses that *seem* authoritative and helpful, *regardless of truth*
- This can result in making up facts + hallucinations

**Models** of human preferences are *even more* unreliable!



# Remove RL from RLHF

**Direct Preference Optimization:**
**Your Language Model is Secretly a Reward Model**


**Rafael Rafailov**[*†] **Archit Sharma**[*†] **Eric Mitchell**[*†]

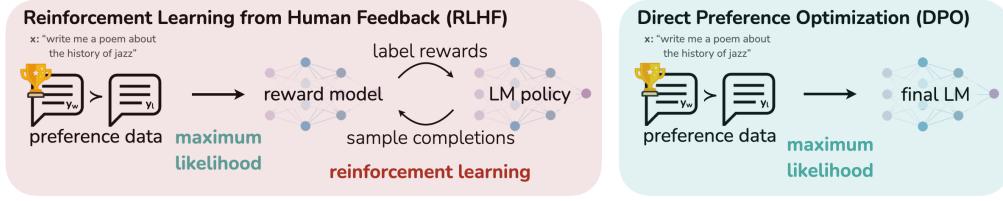
**Stefano Ermon**[†‡] **Christopher D. Manning**[†] **Chelsea Finn**[†]

[†]Stanford University [‡]CZ Biohub
{rafailov,architsh,eric.mitchell}@cs.stanford.edu



**Abstract**

While large-scale unsupervised language models (LMs) learn broad world knowledge and some reasoning skills, achieving precise control of their behavior is difficult due to the completely unsupervised nature of their training. Existing methods for gaining such steerability collect human labels of the relative quality of model generations and fine-tune the unsupervised LM to align with these preferences, often with reinforcement learning from human feedback (RLHF). However, RLHF is a complex and often unstable procedure, first fitting a reward model that reflects the human preferences, and then fine-tuning the large unsupervised LM using reinforcement learning to maximize this estimated reward without drifting too far from the original model. In this paper we introduce a new parameterization of the reward model in RLHF that enables extraction of the corresponding optimal policy in closed form, allowing us to solve the standard RLHF problem with only a simple classification loss. The resulting algorithm, which we call *Direct Preference Optimization* (DPO), is stable, performant, and computationally lightweight, eliminating the need for sampling from the LM during fine-tuning or performing significant hyperparameter tuning. Our experiments show that DPO can fine-tune LMs to align with human preferences as well as or better than existing methods. Notably, fine-tuning with DPO exceeds PPO-based RLHF in ability to control sentiment of generations, and matches or improves response quality in summarization and single-turn dialogue while being substantially simpler to implement and train.


$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$

$$-\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Bigg[\underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when reward estimate is wrong}}\Bigg[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l}\Bigg]\Bigg]$$

$$\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$$

# Test-time Scaling

## Large Language Monkeys: Scaling Inference Compute with Repeated Sampling

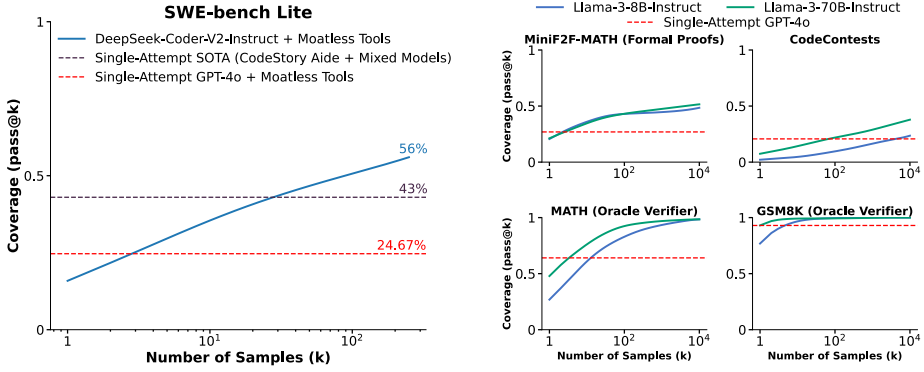
Bradley Brown*†‡, Jordan Juravsky*†, Ryan Ehrlich*†, Ronald Clark‡, Quoc V. Le§, Christopher Ré†, and Azalia Mirhoseini†§

†Department of Computer Science, Stanford University
‡University of Oxford
§Google DeepMind

bradley.brown@cs.ox.ac.uk, jbj@stanford.edu, ryanehrlich@cs.stanford.edu, ronald.clark@cs.ox.ac.uk, qvl@google.com, chrismre@stanford.edu, azalia@stanford.edu

**Abstract**

Scaling the amount of compute used to train language models has dramatically improved their capabilities. However, when it comes to inference, we often limit models to making only one attempt at a problem. Here, we explore inference compute as another axis for scaling, using the simple technique of repeatedly sampling candidate solutions from a model. Across multiple tasks and models, we observe that coverage – the fraction of problems that are solved by any generated sample – scales with the number of samples over four orders of magnitude. Interestingly, the relationship between coverage and the number of samples is often log-linear and can be modelled with an exponentiated power law, suggesting the existence of inference-time scaling laws. In domains like coding and formal proofs, where answers can be automatically verified, these increases in coverage directly translate into improved performance. When we apply repeated sampling to SWE-bench Lite, the fraction of issues solved with DeepSeek-Coder-V2-Instruct increases from 15.9% with one sample to 56% with 250 samples, outperforming the single-sample state-of-the-art of 43%. In domains without automatic verifiers, we find that common methods for picking from a sample collection (majority voting and reward models) plateau beyond several hundred samples and fail to fully scale with the sample budget.




## Scaling LLM Test-Time Compute Optimally Can be More Effective than Scaling Parameters for Reasoning

**Charlie Snell*, Jaehoon Lee§, Kelvin Xu§†, Aviral Kumar#§†**

**Abstract**

Enabling LLMs to improve their outputs by using more test-time compute is a critical step towards building self-improving agents that can operate on open-ended natural language. In this paper, we scale up inference-time computation in LLMs, with a focus on answering: *if an LLM is allowed to use a fixed but non-trivial amount of inference-time compute, how much can it improve its performance on a challenging prompt?* Answering this question has implications not only on performance, but also on the future of LLM pretraining and how to tradeoff inference-time and pre-training compute. Little research has attempted to understand the scaling behaviors of test-time inference methods, with current work largely pro-

# Reasoning Models

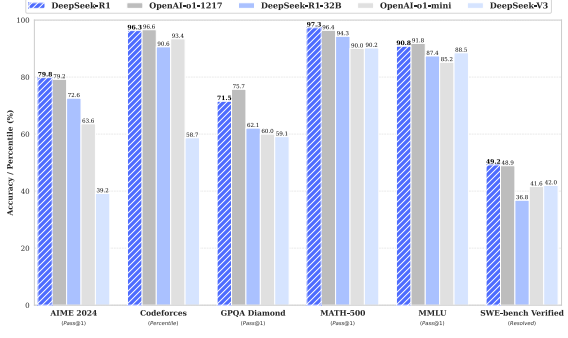
**DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning**

DeepSeek-AI

research@deepseek.com

**Abstract**

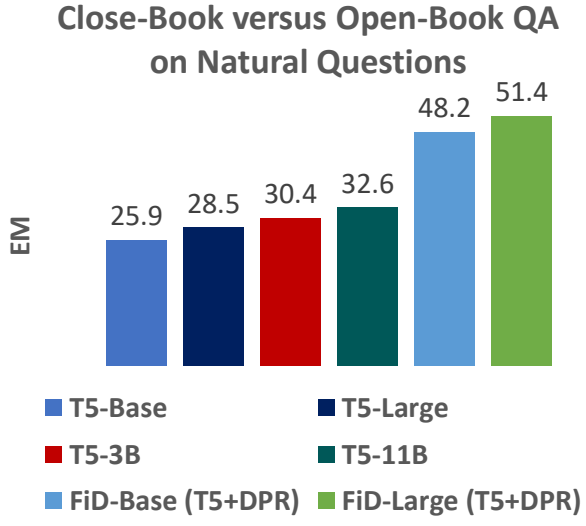
We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Qwen and Llama.




(1) Finetune with Long-CoT data
(2) GRPO + rule-based reward
(3) Distillation

# Retrieval-Augmented Pretraining

LLMs can memorize lots of knowledge in its parameters

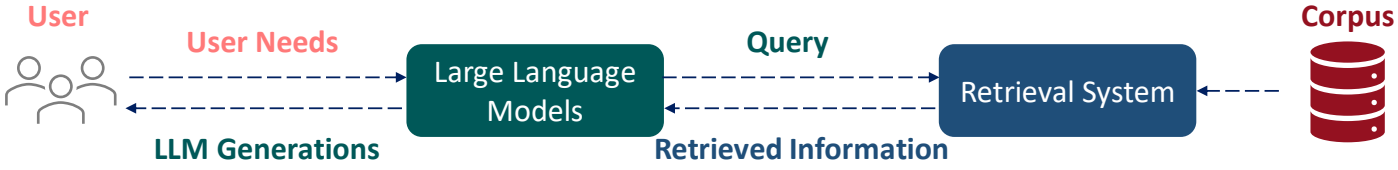



Clear benefits from adding external information from a retrieval system on knowledge-intensive tasks

- Better overall accuracy
- Significantly better parameter efficiency
- More reliable/explicit source of information

# Retrieval Augmentation

- Can we alleviate LLMs from costly parametric memory by augmenting them with a retrieval system?
  - Retrieval system serves as an external memory
  - LLMs learn to leverage retrieved information
- Ideally:
  - More efficient parameter usage
  - Better generalization ability by switching external memory (retrieval corpus)
  - Clear separation of memorization and understanding for transparency and control



# KNN-LM

## GENERALIZATION THROUGH MEMORIZATION: NEAREST NEIGHBOR LANGUAGE MODELS

**Urvashi Khandelwal**[†]**; Omer Levy**[‡]**, Dan Jurafsky**[†]**, Luke Zettlemoyer**[‡] **& Mike Lewis**[‡]
[†]Stanford University
[‡]Facebook AI Research
{urvashik,jurafsky}@stanford.edu
{omerlevy,lsz,mikelewis}@fb.com

### ABSTRACT

We introduce *k*NN-LMs, which extend a pre-trained neural language model (LM) by linearly interpolating it with a $k$-nearest neighbors ($k$NN) model. The nearest neighbors are computed according to distance in the pre-trained LM embedding space, and can be drawn from any text collection, including the original LM training data. Applying this augmentation to a strong WIKITEXT-103 LM, with neighbors drawn from the original training set, our *k*NN-LM achieves a new state-of-the-art perplexity of 15.79 – a 2.9 point improvement with no additional training. We also show that this approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore, again without further training. Qualitatively, the model is particularly helpful in predicting rare patterns, such as factual knowledge. Together, these results strongly suggest that learning similarity between sequences of text is easier than predicting the next word, and that nearest neighbor search is an effective approach for language modeling in the long tail.

### 1 INTRODUCTION

Neural language models (LMs) typically solve two subproblems: (1) mapping sentence prefixes to fixed-sized representations, and (2) using these representations to predict the next word in the text (Bengio et al., 2003; Mikolov et al., 2010). We present a new language modeling approach that is based on the hypothesis that the representation learning problem may be easier than the prediction problem. For example, any English speaker knows that *Dickens is the author of* and *Dickens wrote* will have essentially the same distribution over the next word, even if they do not know what that distribution is. We provide strong evidence that existing language models, similarly, are much better at the first problem, by using their prefix embeddings in a simple nearest neighbor scheme that significantly improves overall performance.

KNN-LM: Interpolate LM prediction with a k-nearest neighbor (KNN) model

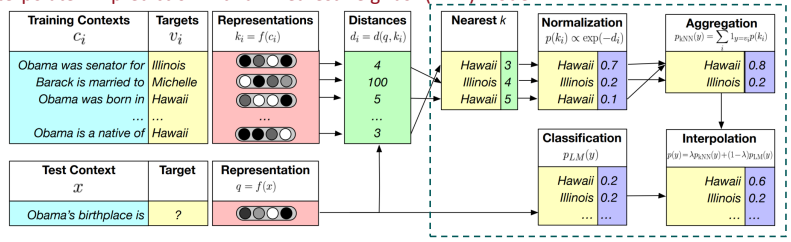



Figure 1: The pipeline of KNN-LM [2]

**When** to retrieve?

- Every token position at inference time

**What** to retrieve?

- (Context, Target Word) pairs: $(c_i, v_i)$
- Key: $k = f(c_i)$ the hidden representation of context from the LM
- Value: the actual ground truth word for the context

# KNN-LM

## GENERALIZATION THROUGH MEMORIZATION: NEAREST NEIGHBOR LANGUAGE MODELS

**Urvashi Khandelwal**[†]**, Omer Levy**[‡]**, Dan Jurafsky**[†]**, Luke Zettlemoyer**[‡] **& Mike Lewis**[‡]
[†]Stanford University
[‡]Facebook AI Research
{urvashik,jurafsky}@stanford.edu
{omerlevy,lsz,mikelewis}@fb.com

### ABSTRACT

We introduce *k*NN-LMs, which extend a pre-trained neural language model (LM) by linearly interpolating it with a $k$-nearest neighbors ($k$NN) model. The nearest neighbors are computed according to distance in the pre-trained LM embedding space, and can be drawn from any text collection, including the original LM training data. Applying this augmentation to a strong WIKITEXT-103 LM, with neighbors drawn from the original training set, our *k*NN-LM achieves a new state-of-the-art perplexity of 15.79 – a 2.9 point improvement with no additional training. We also show that this approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore, again without further training. Qualitatively, the model is particularly helpful in predicting rare patterns, such as factual knowledge. Together, these results strongly suggest that learning similarity between sequences of text is easier than predicting the next word, and that nearest neighbor search is an effective approach for language modeling in the long tail.

### 1 INTRODUCTION

Neural language models (LMs) typically solve two subproblems: (1) mapping sentence prefixes to fixed-sized representations, and (2) using these representations to predict the next word in the text (Bengio et al., 2003; Mikolov et al., 2010). We present a new language modeling approach that is based on the hypothesis that the representation learning problem may be easier than the prediction problem. For example, any English speaker knows that *Dickens is the author of* and *Dickens wrote* will have essentially the same distribution over the next word, even if they do not know what that distribution is. We provide strong evidence that existing language models, similarly, are much better at the first problem, by using their prefix embeddings in a simple nearest neighbor scheme that significantly improves overall performance.

KNN-LM: Interpolate LM prediction with a k-nearest neighbor (KNN) model

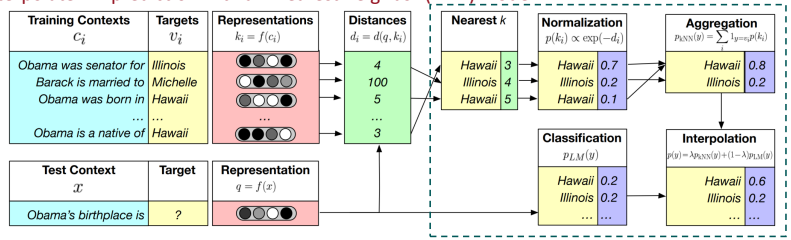



Figure 1: The pipeline of KNN-LM [2]

**How** to retrieve?

- K nearest neighbor search using current context $x$
- Query: q $= f(x)$ the hidden representation of the inference context
- Retrieval Function: standard KNN search with L2 distance

**Where** to retrieve?

- The training corpus with the pretrained LM to formulate the key-value pairs
- Or plug-in a new corpus to generalize to the new domain. E.g. Wiki→Book

# KNN-LM

## Generalization through Memorization: Nearest Neighbor Language Models

**Urvashi Khandelwal**[†]**, Omer Levy**[‡]**, Dan Jurafsky**[†]**, Luke Zettlemoyer**[‡] **& Mike Lewis**[‡]
[†]Stanford University
[‡]Facebook AI Research
{urvashik,jurafsky}@stanford.edu
{omerlevy,lsz,mikelewis}@fb.com

### Abstract

We introduce *k*NN-LMs, which extend a pre-trained neural language model (LM) by linearly interpolating it with a *k*-nearest neighbors (*k*NN) model. The nearest neighbors are computed according to distance in the pre-trained LM embedding space, and can be drawn from any text collection, including the original LM training data. Applying this augmentation to a strong WIKITEXT-103 LM, with neighbors drawn from the original training set, our *k*NN-LM achieves a new state-of-the-art perplexity of 15.79 – a 2.9 point improvement with no additional training. We also show that this approach has implications for efficiently scaling up to larger training sets and allows for effective domain adaptation, by simply varying the nearest neighbor datastore, again without further training. Qualitatively, the model is particularly helpful in predicting rare patterns, such as factual knowledge. Together, these results strongly suggest that learning similarity between sequences of text is easier than predicting the next word, and that nearest neighbor search is an effective approach for language modeling in the long tail.

### 1 Introduction

Neural language models (LMs) typically solve two subproblems: (1) mapping sentence prefixes to fixed-sized representations, and (2) using these representations to predict the next word in the text (Bengio et al., 2003; Mikolov et al., 2010). We present a new language modeling approach that is based on the hypothesis that the representation learning problem may be easier than the prediction problem. For example, any English speaker knows that *Dickens is the author of* and *Dickens wrote* will have essentially the same distribution over the next word, even if they do not know what that distribution is. We provide strong evidence that existing language models, similarly, are much better at the first problem, by using their prefix embeddings in a simple nearest neighbor scheme that significantly improves overall performance.

KNN-LM: Interpolate LM prediction with a k-nearest neighbor (KNN) model

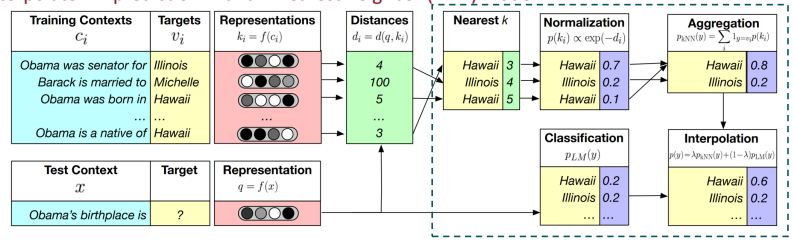



Figure 1: The pipeline of KNN-LM [2]

**How** to leverage retrieved information?

- Normalize and aggregate retrieved scores to obtain the KNN output
- Linearly interpolate the original LM output with KNN output

$$p(y) = \lambda p_{\text{knn}}(y) + (1-\lambda) p_{\text{lm}}(y)$$

# REALM

**REALM: Retrieval-Augmented Language Model Pre-Training**

**Kelvin Guu** [*1] **Kenton Lee** [*1] **Zora Tung** [1] **Panupong Pasupat** [1] **Ming-Wei Chang** [1]

**Abstract**

Language model pre-training has been shown to capture a surprising amount of world knowledge, crucial for NLP tasks such as question answering. However, this knowledge is stored implicitly in the parameters of a neural network, requiring ever-larger networks to cover more facts. To capture knowledge in a more modular and interpretable way, we augment language model pre-training with a latent *knowledge retriever*, which allows the model to retrieve and attend over documents from a large corpus such as Wikipedia, used during pre-training, fine-tuning and inference. For the first time, we show how to pre-train such a knowledge retriever in an unsupervised manner, using masked language modeling as the learning signal and backpropagating through a retrieval step that considers millions of documents. We demonstrate the effectiveness of Retrieval-Augmented Language Model pre-training (REALM) by fine-tuning on the challenging task of Open-domain Question Answering (Open-QA). We compare against state-of-the-art models for both explicit and implicit knowledge storage on three popular Open-QA benchmarks, and find that we outperform all previous methods by a significant margin (4-16% absolute accuracy), while also providing qualitative benefits such as interpretability and modularity.

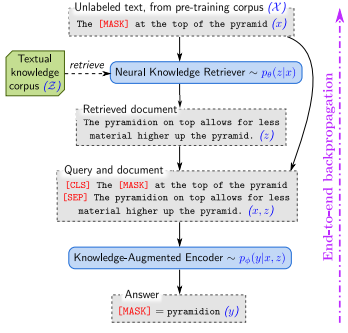



*Figure 1.* REALM augments language model pre-training with a **neural knowledge retriever** that retrieves knowledge from a **textual knowledge corpus**, $\mathcal{Z}$ (e.g., all of Wikipedia). Signal from the language modeling objective backpropagates all the way through the retriever, which must consider millions of documents in $\mathcal{Z}$—a significant computational challenge that we address.

correctly predict the missing word in the following sentence: "The ___ is the currency of the United Kingdom" (answer: "pound").

**When** to retrieve?
- Once every training/testing sequence

**What** to retrieve?
- Similar text sequences

**How** to retrieve?
- Dense retriever (BERT here) using current sequence as the query

**Where** to retrieve?
- The same pretraining corpus

**How** to leverage retrieved information?
- Add retrieved sequence as extra inputs
- Pretrain LM to learn how to use these extra information (hopefully)

# Retro

## Improving language models by retrieving from trillions of tokens

**Sebastian Borgeaud[†], Arthur Mensch[†], Jordan Hoffmann[†], Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae[‡], Erich Elsen[‡] and Laurent Sifre[†,‡]**

All authors from DeepMind, [†]Equal contributions, [‡]Equal senior authorship

**We enhance auto-regressive language models by conditioning on document chunks retrieved from a large corpus, based on local similarity with preceding tokens. With a 2 trillion token database, our Retrieval-Enhanced Transformer (RETRO) obtains comparable performance to GPT-3 and Jurassic-1 on the Pile, despite using 25× fewer parameters. After fine-tuning, RETRO performance translates to downstream knowledge-intensive tasks such as question answering. RETRO combines a frozen BERT retriever, a differentiable encoder and a chunked cross-attention mechanism to predict tokens based on an order of magnitude more data than what is typically consumed during training. We typically train RETRO from scratch, yet can also rapidly RETROfit pre-trained transformers with retrieval and still achieve good performance. Our work opens up new avenues for improving language models through explicit memory at unprecedented scale.**

### 1. Introduction

Language modelling (LM) is an unsupervised task that consists of modelling the probability of text, usually by factorising it into conditional next-token predictions $p(x_1, \ldots, x_n) = \prod_i p(x_i|x_{<i})$. Neural networks have proven to be powerful language models, first in the form of recurrent architectures (Graves, 2013; Jozefowicz et al., 2016; Mikolov et al., 2010) and more recently in the form of Transformers (Vaswani et al., 2017), that use attention to contextualise the past. Large performance improvements have come from increasing the amount of data, training compute, or model parameters. Transformers have been scaled from 100 million parameter models in seminal work to over hundred billion parameters (Brown et al., 2020; Radford et al., 2019) in the last two years which has led to models that do very well on a wide array of tasks in a zero or few-shot formulation. Increasing model

**When** to retrieve?
- Split a pretraining sequence into chunks (e.g. 64 tokens per chunk)
- Retrieve similar chunks for each chunk

**What** to retrieve?
- Forming key-value pairs as (this chunk, next chunk) from documents of the corpus
- Retrieve key chunks (x), augment value chunk (y).

**How** to retrieve?
- Represent chunks by average BERT embeddings across its token positions
- Retrieval by L2 distance in their representations from ANNS index
- Use SCANN to enable 10 million second latency per querying from 2 trillion tokens (31 Billion Embeddings)

**Where** to retrieve?
- All chunks from the pretraining corpus, embedded by frozen BERTs
- Again, can switch the retrieval corpus at inference time

**How** to leverage retrieved information
- New Chunked Cross-Attention (CCA) blocks: each chunk attends to retrieved chunks of the previous chunk
- Inter leaving CCA blocks and normal decoder attention blocks in each RETRO attention layer
- Pretraining end-to-end at large scale

# Retrieve In-context Examples

**Learning To Retrieve Prompts for In-Context Learning**

**Ohad Rubin** **Jonathan Herzig** **Jonathan Berant**
The Blavatnik School of Computer Science, Tel Aviv University
{ohad.rubin,jonathan.herzig,joberant}@cs.tau.ac.il

**Abstract**

*In-context learning* is a recent paradigm in natural language understanding, where a large pre-trained language model (LM) observes a test instance and a few training examples as its input, and directly decodes the output without any update to its parameters. However, performance has been shown to strongly depend on the selected training examples (termed *prompts*). In this work, we propose an efficient method for retrieving prompts for in-context learning using annotated data and an LM. Given an input-output pair, we estimate the probability of the output given the input and a candidate training example as the prompt, and label training examples as positive or negative based on this probability. We then train an efficient dense retriever from this data, which is used to retrieve training examples as prompts at test time. We evaluate our approach on three sequence-to-sequence tasks where language utterances are mapped to meaning representations, and find that it sub-

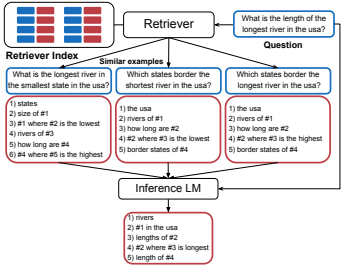



Figure 1: An overview of prompt retrieval: Given a question from BREAK, one retrieves similar training examples from an index of the training set. The question and training examples (the prompt) are passed to an inference LM that decodes the output.

(2021a) showed that downstream performance can

**When** to retrieve?
- Per downstream data point

**What** to retrieve?
- Similar training data points (x, y)

**How** to retrieve?
- Dense retriever adapted for LLM

**Where** to retrieve?
- Training data

**How** to leverage retrieved information?
- Add in as in-context examlpes

# Retrieve In-context Examples

**Learning To Retrieve Prompts for In-Context Learning**

**Ohad Rubin** **Jonathan Herzig** **Jonathan Berant**
The Blavatnik School of Computer Science, Tel Aviv University
{ohad.rubin,jonathan.herzig,joberant}@cs.tau.ac.il

**Abstract**

*In-context learning* is a recent paradigm in natural language understanding, where a large pre-trained language model (LM) observes a test instance and a few training examples as its input, and directly decodes the output without any update to its parameters. However, performance has been shown to strongly depend on the selected training examples (termed *prompts*). In this work, we propose an efficient method for retrieving prompts for in-context learning using annotated data and an LM. Given an input-output pair, we estimate the probability of the output given the input and a candidate training example as the prompt, and label training examples as positive or negative based on this probability. We then train an efficient dense retriever from this data, which is used to retrieve training examples as prompts at test time. We evaluate our approach on three sequence-to-sequence tasks where language utterances are mapped to meaning representations, and find that it sub-

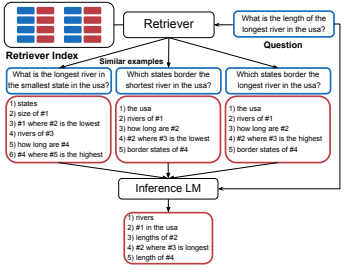



Figure 1: An overview of prompt retrieval: Given a question from BREAK, one retrieves similar training examples from an index of the training set. The question and training examples (the prompt) are passed to an inference LM that decodes the output.

(2021a) showed that downstream performance can

**When** does this work?

- A generally better way to fine more similar in-context examples than random sample

**Why** does this work?

- A form of test time learning

**Test Time Learning**

- Find similar training data points for the current testing data point
- Focused learning (e.g., a few gradient steps) on similar training data points
- “Upweighting” training data close to the current testing data
- A classic idea

**“In-context learning == SGD view”**

- Performing virtual SGD on random versus retrieved similar in-context examples

# Retrieve In-context Examples

**Learning To Retrieve Prompts for In-Context Learning**

**Ohad Rubin** **Jonathan Herzig** **Jonathan Berant**
The Blavatnik School of Computer Science, Tel Aviv University
{ohad.rubin,jonathan.herzig,joberant}@cs.tau.ac.il

**Abstract**

*In-context learning* is a recent paradigm in natural language understanding, where a large pre-trained language model (LM) observes a test instance and a few training examples as its input, and directly decodes the output without any update to its parameters. However, performance has been shown to strongly depend on the selected training examples (termed *prompts*). In this work, we propose an efficient method for retrieving prompts for in-context learning using annotated data and an LM. Given an input-output pair, we estimate the probability of the output given the input and a candidate training example as the prompt, and label training examples as positive or negative based on this probability. We then train an efficient dense retriever from this data, which is used to retrieve training examples as prompts at test time. We evaluate our approach on three sequence-to-sequence tasks where language utterances are mapped to meaning representations, and find that it sub-

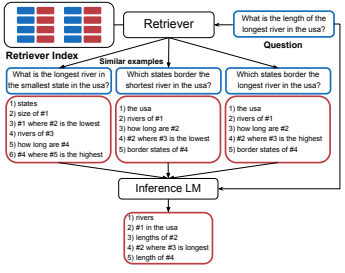



Figure 1: An overview of prompt retrieval: Given a question from BREAK, one retrieves similar training examples from an index of the training set. The question and training examples (the prompt) are passed to an inference LM that decodes the output.

(2021a) showed that downstream performance can

**When** does this work?

- A generally better way to fine more similar in-context examples than random sample

**Why** does this work?

- A form of test time learning

**Test Time Learning**

- Find similar training data points for the current testing data point
- Focused learning (e.g., a few gradient steps) on similar training data points
- “Upweighting” training data close to the current testing data
- A classic idea

**“In-context learning == SGD view”**

- Performing virtual SGD on random versus retrieved similar in-context examples

# Retrieve Extra Knowledge

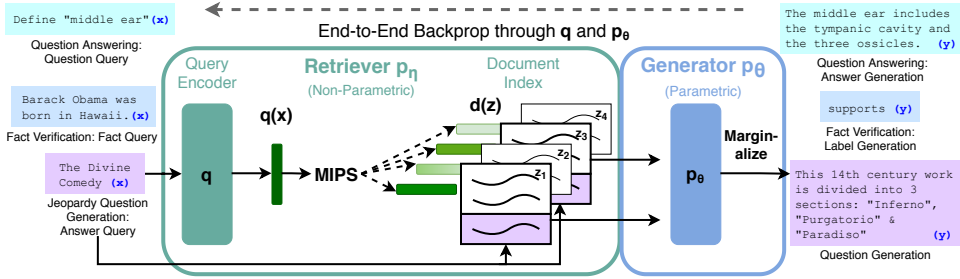
**Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks**

**Patrick Lewis**[†‡], **Ethan Perez**[⋆], **Aleksandra Piktus**[†], **Fabio Petroni**[†], **Vladimir Karpukhin**[†], **Naman Goyal**[†], **Heinrich Küttler**[†], **Mike Lewis**[†], **Wen-tau Yih**[†], **Tim Rocktäschel**[†‡], **Sebastian Riedel**[†‡], **Douwe Kiela**[†]

[†]Facebook AI Research; [‡]University College London; [⋆]New York University;
plewis@fb.com

**Abstract**

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages




**When** to retrieve?
- Per downstream data point

**What** to retrieve?
- Relevant documents (x)

**How** to retrieve?
- Dense retriever, e.g., from web search

**Where** to retrieve?
- Target corpus with needed information

**How** to leverage retrieved information?
- Used to be complex:
  - Latent space models
  - Fusion-in-Decoder
- With Decoder LLM:
  - As additional inputs (with prompts)
  - Zero-shot or finetuned

**When** does this work?
- Tasks required additional information
- E.g., Knowledge-intensive tasks, reducing hallucination, plug-in in-domain information, etc.

**Why** does it work?
- Additional information from retrieval
- An LLM version of OpenQA

**When** does it **not** work?
- Tasks do not require extra information
- E.g., hard to think of what to extra information is needed for sentiment analysis or grammar check